

# Random Sampling of Latin squares via binary contingency tables and probabilistic divide-and-conquer

Stephen DeSalvo<sup>1</sup>

March 24, 2017

*Abstract.* We demonstrate a novel approach for the random sampling of Latin squares of order  $n$  via probabilistic divide-and-conquer. The algorithm divides the entries of the table modulo powers of 2, and samples a corresponding binary contingency table at each level. The sampling distribution is based on the Boltzmann sampling heuristic, along with probabilistic divide-and-conquer.

## 1 Introduction

### 1.1 Motivation

When tasked with random sampling from a complicated state space, a popular choice is to run a rapidly mixing Markov chain. Markov chains have proven to be an incredibly powerful and versatile tool, however, there are several reasons why one would consider alternative methods. In particular, unless the chain starts off in stationarity, or has been coupled from the past [55], the error is seldom zero after a finite number of steps; quite often this is an acceptable tradeoff, and through a more complicated analysis one can bound the sampling error. For sampling problems with a large number of constraints, however, it is often difficult to analyze the resulting Markov chain and prove mixing time bounds. In addition, there is no general scheme to parallelize the sequential steps in a Markov chain, and many naive approaches can lead to unintended sampling bias [59, Chapter 18].

Another paradigm of random sampling, popular in combinatorics, is the Boltzmann sampler [31, 36], where one samples from some combinatorial structure via independent random variables, determined by the form of the generating function to closely resemble the true distribution of component sizes in the random combinatorial structure. The result is a random combinatorial object with a random set of statistics, referred to as a *grand canonical ensemble* in the statistical mechanics literature. For a fixed set of conditions, an exact sample can be obtained by sampling repeatedly until all of the conditions are satisfied, throwing away samples which miss the target [30]. We have the structure which is typical of Boltzmann sampling, which is a collection of independent random variables subject to a condition. However, unlike the plethora of combinatorial structures for which a randomly generated structure with random statistics contains an acceptable amount of bias, there are good reasons to demand that all statistics are satisfied, see for example [15, 54]. Fortunately, owing to the large amount of independence in the formulation, Boltzmann samplers are embarrassingly parallel, offering many different effective means of parallelizing the computation. Unfortunately, as with many multivariate combinatorial structures, the rejection costs associated with waiting until all statistics match exactly are prohibitively large, even with effective parallelizing.

The Boltzmann sampler has been described as a generalization of the table methods of Nijenhuis and Wilf, also known as the *recursive method*, see [51], although from our point of view it has a distinctly different flavor. The recursive method champions creating a table of numerical values

---

<sup>1</sup>Department of Mathematics, University of California Los Angeles. stephendesalvo@math.ucla.edu

which can be used to generate individual components of a sample in its correct proportion in an unbiased sample. This approach, however, demands the computing of a lookup table, which can be both prohibitively large in size and prohibitively long to compute each entry, typically by a recursion.

Our approach lies somewhere in between the recursive method and Boltzmann sampling, with a distinctly probabilistic twist, and which avoids Markov chains altogether. Our approach, *probabilistic divide-and-conquer* (PDC) [1], provides an object which satisfies all constraints, and which targets the marginal distribution of components, in this case entries in a table, according to the Boltzmann sampling principle. Rather than sample all entries of the table at once and apply a rejection function which is either 0 or 1, we instead sample one bit of each entry in the table, one at a time, essentially building the table via its bits, starting with the least significant bit. In the case where rejection sampling probabilities are known or can be computed to arbitrary accuracy, the algorithm is unbiased. Probabilistic divide-and-conquer is similar in many respects to the recursive method, by approaching the problem in pieces and selecting a piece in proportion to already observed pieces. However, whereas the recursive method typically involves a more geometric/spatial decomposition of a finite, discrete sample space, we instead champion a probabilistic decomposition of the sample space, by decomposing the random variables themselves which describe random component-sizes. This approach also generalizes in a straightforward manner to sampling from continuous sample spaces and lower dimensional subspaces, see [23], as long as the random variables can be decomposed effectively.

In the remainder of this section, we define  $(r, c)$ -contingency tables and Latin squares of order  $n$ , and describe some of the standard algorithms to randomly sample them. Section 1.3 contains the highest-level explanation of our proposed algorithm for Latin squares of order  $n$ , and in Section 2 we present the probabilistic tools which justify this approach. Section 3 contains the complete statements of the algorithms, along with a word of caution in the final subsection.

## 1.2 $(r, c)$ -contingency tables

**Definition 1.1.** An  $(r, c)$ -contingency table is an  $m$  by  $n$  table of nonnegative integer values with row sums given by  $r = (r_1, \dots, r_m)$  and column sums given by  $c = (c_1, \dots, c_n)$ . A table for which the entries are further assumed to be in the set  $\{0, 1\}$  is called a *binary  $(r, c)$ -contingency table*.

The exact, uniform sampling from the set of  $(r, c)$ -contingency tables is a well-studied problem. Owing to the large parameter space, there are a plethora of results spanning many decades pertaining just to counting the number of such tables, see for example [5, 10, 56, 2, 21, 3, 4, 41, 6, 9, 8, 27, 11, 53, 40]. There is also interest in various Markov chain approaches, including coupling from the past, see for example [15, 18, 19, 29, 35, 32, 48, 46, 14].

The random sampling of contingency tables is an extensive topic, and we do not attempt to recount all previously known results in this area, and instead refer the interested reader to, e.g., the introduction in [19] and the references therein. As mentioned previously, the main sampling approach traditionally championed is to use a Markov chain, e.g., starting with the general framework in [29], to define an appropriate state transition matrix, and then prove that such a chain is rapidly mixing; indeed, this approach has been profoundly fruitful, see for example [32, 48, 46]. Explicitly, one state transition championed in [27] is to sample two ordered rows and two ordered columns uniformly at

random, and, *if possible*, apply the transformation to their entries

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \rightarrow \begin{pmatrix} a_{11} + 1 & a_{12} - 1 \\ a_{21} - 1 & a_{22} + 1 \end{pmatrix}.$$

If such a transformation would force the table to lie outside the solution set, then a different pair of ordered rows and columns is generated. This chain was shown in [27] to be ergodic and converge to the uniform distribution over the set of contingency tables. Mixing times were later proved in [28, 42, 16] in various contexts. Other Markov chains have also been proposed, see for example [32, 19]. The recent book [44] also contains many more examples involving Markov chains and coupling from the past to obtain exact samples.

An arguably unique approach to random sampling of contingency tables is contained in [33]. The algorithm associates to each state a parallelepiped with respect to some basis, and defines some convex set which contains all of the parallelepipeds. Then, one samples from this convex set, and if the sample generated lies within one of the parallelepipeds, it returns the corresponding state; otherwise, restart. This algorithm was shown to be particularly effective in [49] when the row sums are all  $\Omega(n^{3/2}m \log m)$  and the column sums are all  $\Omega(m^{3/2}n \log n)$ .

Recently, a self-similar PDC algorithm was championed for the random sampling of contingency tables [26], offering an arguably new approach to random sampling of these intricate structures. In related work, the author demonstrated how to utilize PDC to improve upon existing algorithms for exact random sampling of Latin squares of order  $n$  [24]. The current work, motivated by improving further still the random sampling of Latin squares of order  $n$ , extends the original PDC sampling algorithm for contingency tables to the more general case when certain entries are forced to be 0, see for example [13, 12]. The probabilistic analysis is straightforward, although the computational complexity changes drastically. Nevertheless, our numerical experiments demonstrate that this approach is simple, practical and executes fairly rapidly for a large class of tables. Thus, we champion our approach for practitioners looking for simple, alternative methods for sampling quickly from these intricate structures.

### 1.3 Latin squares of order $n$

**Definition 1.2.** A Latin square of order  $n$  is an  $n \times n$  table of values such that the numbers from the set  $\{1, 2, \dots, n\}$  appear exactly once in each row and in each column.

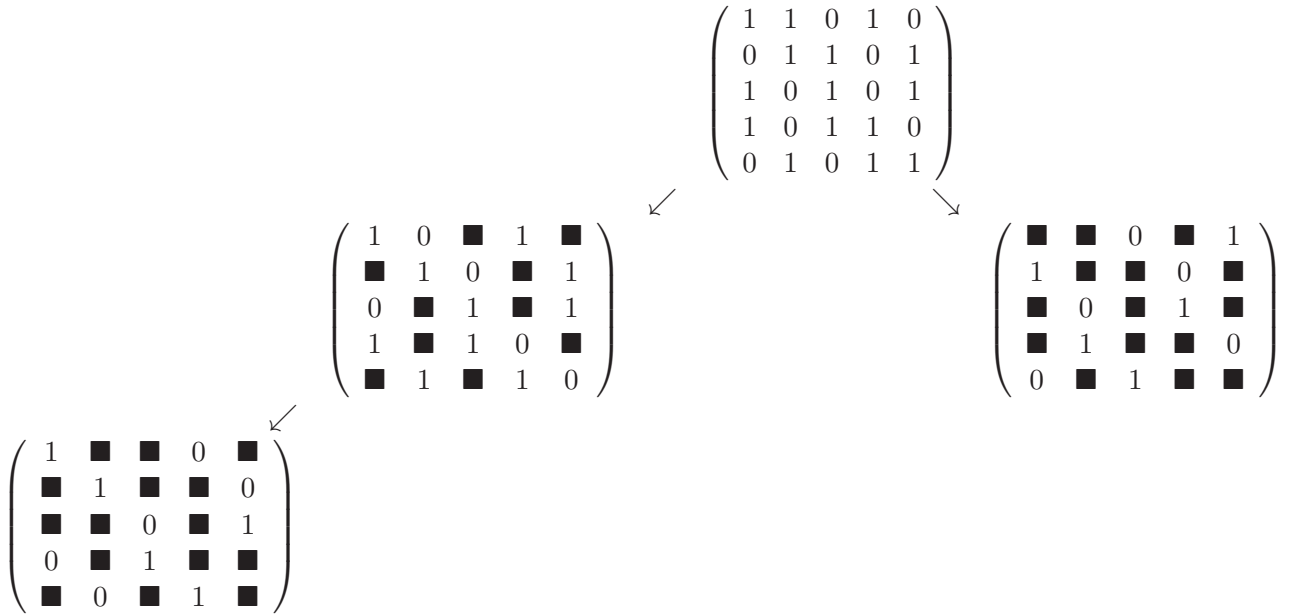
There are many techniques available for random sampling of Latin squares of order  $n$ , see for example [24, Section 6]. A Markov chain approach is contained in [45]; in particular, they construct and analyze two different Markov chains. The transition states are described via an explicit algorithm, requiring  $O(n)$  time to transition from one state to the next, and guaranteed to satisfy all of the Latin square constraints. However, as far as we are aware, even though the stationary distribution was proved to be uniform over the set of Latin squares of order  $n$ , neither of the Markov chains presented have been shown to be rapidly mixing. Another approach is to decompose the entries of a Latin square as sums of mutually disjoint permutation matrices, as in [20, 38, 37, 60]. This approach has practical limitations, as the straightforward rejection algorithm is prohibitive, and the more involved approach in [38] requires an auxiliary computation which dominates the cost of the algorithm. We suspect that this approach may benefit from a probabilistic divide-and-conquer approach, though we have not carried out the relevant analysis.

An informal description of our algorithm for random sampling of Latin squares of order  $n$  is as follows; see Algorithm 6 for the complete description. For simplicity of exposition, we assume  $n = 2^m$  for some positive integer  $m$ , which has no theoretical significance, but avoids cumbersome notation involving rounding. First, sample an  $n \times n$  binary contingency table with all row sums and column sums equal to  $n/2$ . Then consider the subset of entries in which a 1 appears, of which there are  $n/2$  in each row and column, and sample within that subset of entries an  $(n/2) \times (n/2)$  binary contingency table with all row sums and column sums equal to  $n/4$ ; do the same for the subset of entries in which a 0 appears. Repeat this process through all  $m$  levels. By interpreting the sequence of  $m$  1s and 0s as the binary expansion of a positive integer, we obtain a Latin square of order  $n$ .

This idea, that of generating a Latin square by its bits, could be considered a straightforward divide-and-conquer algorithm if one simply desires an object from the solution set. What makes our treatment more intricate is our approximation heuristic, designed to target the uniform distribution over the solution set; see Section 2.3.

Of course, in order to avoid bias in the sampling algorithm, we must sample each of the  $n/2^\ell \times n/2^\ell$  binary contingency tables in their correct proportion of partially completed Latin squares of order  $n$ ; this is not such a trivial task, and requires either counting all possible completions, a computationally extensive task, or a more direct analysis of the resulting multivariate probability governing the acceptance probability, given explicitly in Equation (9). While both approaches present technical difficulties, the probabilistic formulation yields a natural approximation heuristic, which we present in Equation (10), which is computable in polynomial time. It assumes independence of columns, a reasonable approximation heuristic in many parameter spaces of interest, while still enforcing several necessary conditions. A more complete probabilistic description of our heuristic is contained in Section 2.3.

**Example 1.1.** The following is an example of how to apply the algorithm described above.



This corresponds to the following order 5 Latin square

$$\begin{pmatrix} 5 & 1 & 2 & 3 & 4 \\ 4 & 5 & 1 & 2 & 3 \\ 1 & 2 & 3 & 4 & 5 \\ 3 & 4 & 5 & 1 & 2 \\ 2 & 3 & 4 & 5 & 1 \end{pmatrix}.$$

**Remark 1.2.** There is an extensive history related to *partially completed Latin squares of order  $n$* , which are Latin squares of order  $n$  for which only some of the  $n^2$  entries have been filled in, and none of those entries a priori violate any of the Latin square conditions. In particular, the treatment in [57, Chapter 17] related to partially completed Latin squares pertains solely to leaving entire entries filled or unfilled, whereas the algorithm described above is of a distinctly different flavor. For example, [57, Theorem 17.1] states that all Latin squares of order  $n$  with their first  $k$  rows filled in can be extended to a partially completed Latin square with the first  $k + 1$  rows filled in, for  $k = 1, 2, \dots, n - 1$ . There are also ways of filling in elements for which no possible Latin square can be realized given those elements; in general, deciding whether a Latin square of order  $n$  with an arbitrary set of squares filled in can be completed is an NP-complete problem, see [17]. It would be interesting to explore the analogous analysis for our proposed bit-by-bit approach.

## 2 Probabilistic approach

### 2.1 Probabilistic divide-and-conquer

Probabilistic divide-and-conquer (PDC) is an approach for exact sampling by dividing up the sample space into two pieces, sampling each one separately, and then piecing them together appropriately [1]. In order to sample using PDC, one starts with a sample space  $\Omega$  decomposed into two separate sets  $\Omega = \mathcal{A} \times \mathcal{B}$ , and writes the target distribution as

$$\mathcal{L}(S) = \mathcal{L}((A, B)|E), \tag{1}$$

where  $A \in \mathcal{A}$  and  $B \in \mathcal{B}$  have given distributions and are independent, and  $E \subset \mathcal{A} \times \mathcal{B}$  is some measurable event of the sample space. Whereas rejection sampling can be described as sampling  $a$  from  $\mathcal{L}(A)$  and  $b$  from  $\mathcal{L}(B)$ , and then checking whether  $(a, b) \in E$  (see Algorithm 1 below), PDC samples  $x$  from  $\mathcal{L}(A|E)$  and  $y$  from  $\mathcal{L}(B|E, a)$ , and returns  $(x, y)$  as an exact sample (see Algorithm 2 below).

---

**Algorithm 1** [1] Standard rejection sampling from  $\mathcal{L}((A, B)|E)$

---

1. Generate sample from  $\mathcal{L}(A)$ , call it  $a$ .
  2. Generate sample from  $\mathcal{L}(B)$ , call it  $b$ .
  3. Check if  $(a, b) \in E$ ; if so, return  $(a, b)$ , otherwise restart.
- 

**Lemma 2.1.** *PDC Lemma [1, Lemma 2.1]. Suppose  $E$  is a measurable event of positive probability. Suppose  $X$  is a random element of  $\mathcal{A}$  with distribution*

$$\mathcal{L}(X) = \mathcal{L}(A|E), \tag{2}$$

---

**Algorithm 2** [1] Probabilistic Divide-and-Conquer sampling from  $\mathcal{L}((A, B)|E)$

---

1. Generate sample from  $\mathcal{L}(A|E)$ , call it  $x$ .
  2. Generate sample from  $\mathcal{L}(B|E, A=x)$  call it  $y$ .
  3. Return  $(x, y)$ .
- 

and  $Y$  is a random element of  $\mathcal{B}$  with conditional distribution

$$\mathcal{L}(Y|X=a) = \mathcal{L}(B|E, A=a). \quad (3)$$

Then  $(X, Y) \stackrel{d}{=} S$ , i.e., the pair  $(X, Y)$  has the same distribution as  $S$ , given by (1).

Often, however, the conditional distributions in Algorithm 2 are not practical to sample from, and so the following PDC algorithm which uses rejection sampling has been proven to be effective and practical in many situations, see for example [23, 25, 24].

---

**Algorithm 3** PDC sampling from  $\mathcal{L}((A, B)|(A, B) \in E)$  using soft rejection sampling

---

1. Generate sample from  $\mathcal{L}(A)$ , call it  $a$ .
  2. Reject  $a$  with probability  $1 - s(a)$ , where  $s(a)$  is a function of  $\mathcal{L}(B)$ ,  $E$ ; otherwise, restart.
  3. Generate sample from  $\mathcal{L}(B|(a, B) \in E)$ , call it  $y$ .
  4. Return  $(a, y)$ .
- 

**Remark 2.2.** A surprisingly efficient division occurs when the sets  $A$ ,  $B$ , and  $E$  are such that  $\mathcal{L}(B|E, A=a)$  is trivial for each  $a \in \mathcal{A}$ . In this case, it was shown in [1, 23], that a nontrivial speedup can be obtained by relatively simple arguments.

**Remark 2.3.** A more recent application in [25] uses the recursive method [51, 52] in order to obtain both the rejection probability  $s(a)$  along with the conditional distributions  $\mathcal{L}(B|(a, B) \in E)$ . It was proved to have a smaller rejection rate than exact Boltzmann sampling, and also requires less computational resources than the full recursive method.

Previous applications of PDC have considered mutually independent random variables  $X_1, X_2, \dots, X_n$ , with target space given by some measurable event  $E$ , with the goal to sample from the distribution

$$\mathcal{L}((X_1, \dots, X_n)|E).$$

We assume that the unconditional distributions  $\mathcal{L}(X_1), \dots, \mathcal{L}(X_n)$  are known and can be sampled. For general two-dimensional tables, we consider sampling problems of the form:  $X_{1,1}, \dots, X_{m,n}$  are mutually independent random variables,  $E$  is some measurable event, and the goal is to sample from the distribution

$$\mathcal{L} \left( \begin{array}{ccc} X_{1,1}, & \dots & X_{1,n}, \\ \vdots & & \vdots \\ X_{m,1}, & \dots & X_{m,n} \end{array} \middle| E \right). \quad (4)$$

The uniform distribution over all  $(r, c)$ -contingency tables can be obtained, see for example [7, 26], by taking  $X_{i,j}$  to be geometrically distributed with parameter  $p_{i,j} = \frac{m}{m+c_j}$ ,  $i = 1, \dots, m$ ,  $j = 1, \dots, n$ , subject to the measurable event

$$E \equiv E_{r,c} = \left\{ \begin{array}{l} \sum_{\ell=1}^n X_{i,\ell} = r_i, \quad \forall 1 \leq i \leq m; \\ \sum_{\ell=1}^m X_{\ell,j} = c_j, \quad \forall 1 \leq j \leq n \end{array} \right\}. \quad (5)$$

Taking  $X_{i,j}$  to be Bernoulli distributed with parameter  $p_{i,j}/(1 + p_{i,j})$ , we obtain the uniform distribution over the set of  $(r, c)$ -binary contingency tables.

In the case of Latin squares of order  $n$ , we consider the parameterization which consists of  $n \times n$  tables of values, each row of which is a permutation of elements  $\{1, 2, \dots, n\}$ , and each column is a permutation of elements  $\{1, 2, \dots, n\}$ . Let  $U_{i,j}$  denote a collection of i.i.d. discrete uniform random variables from the set  $\{1, 2, \dots, n\}$ ,  $1 \leq i, j \leq n$ , and let  $S_n$  denote the set of all permutations of the elements  $\{1, \dots, n\}$ . The sample space for random sampling of Latin squares is then given by Equation (4) with  $\mathcal{L}(X_{i,j}) = \mathcal{L}(U_{i,j})$  and measurable event  $E$  given by

$$E = \left\{ \begin{array}{l} (X_{i,1}, \dots, X_{i,n}) \in S_n, \quad \forall 1 \leq i \leq m; \\ (X_{1,j}, \dots, X_{m,j}) \in S_n, \quad \forall 1 \leq j \leq n \end{array} \right\}.$$

Once this probabilistic framework has been established, the next task is to devise a PDC division that is both practical and amenable to the resulting analysis. We motivate our PDC division in the next section.

## 2.2 An heuristic for contingency tables

We first recall a division from [1] which is recursive and self-similar, and serves as the inspiration for the algorithm championed in this paper. Let  $0 < x < 1$ , and let  $Z_1(x), Z_2(x), \dots, Z_n(x)$  denote independent geometric random variables with  $Z_i \equiv Z_i(x)$  having distribution which is geometric with parameter  $1 - x^i$ ,  $i = 1, 2, \dots, n$ . Conditional on  $\sum_{i=1}^n i Z_i(x) = n$ , the random variables  $Z_i(x)$ ,  $i = 1, 2, \dots, n$ , represent the number of parts of size  $i$  in a uniformly chosen integer partition of size  $n$ ; see [39, 47, 58]. Suppose, for the purpose of this demonstration, that we wish to sample from  $\mathcal{L}((Z_1(x), \dots, Z_n(x)) | \sum_{i=1}^n i Z_i(x) = n)$ .

The division championed in this case is inherently probabilistic, and relies on a property of geometric random variables. Namely, letting  $G(q)$  and  $G'(q^2)$  denote independent geometric random variables with parameters  $1 - q$  and  $1 - q^2$  respectively, and letting  $B(p)$  denote an independent Bernoulli random variable with parameter  $p$ , we have the decomposition

$$G(q) \stackrel{D}{=} B\left(\frac{q}{1+q}\right) + 2G'(q^2), \quad (6)$$

where the equality is in distribution. In other words, this decomposition shows that the bits in a geometric random variable are independent, and specifies their distribution. It also suggests a PDC algorithm; that is, sample the least significant bits first, one at a time, and then the remaining part has a distribution which is two times a geometric distribution with parameter  $q^2$  in place of  $q$ . As a bonus, many sampling algorithms have a tilting parameter which can be chosen arbitrarily, as is the case for parameter  $x$  for integer partitions; after each stage of the PDC algorithm, we may adjust the tilting parameter to more optimally target the remaining sampling problem.

Explicitly, we use the division

$$A = (\epsilon_1, \dots, \epsilon_n), \quad B = (2Z_1(x^2), 2Z_2(x^4), \dots, 2Z_n(x^{2n})),$$

where  $\epsilon_i$  denotes the least significant bit of  $Z_i$ , distributed as a Bernoulli random variable with parameter  $x^i/(1+x^i)$ . Even though we have written the geometric random variables in  $B$  in terms



of a new parameter  $x^2$ , since the distribution we are sampling from is conditionally independent of  $x$ , we are free to choose any  $x' \in (0, 1)$  which more optimally targets the sampling distribution, which is realized after the bits in  $A$  are sampled.

An application of Algorithm 3 in this setting is the following: Sample from  $A = (\epsilon_1, \dots, \epsilon_n)$  and let  $a = \sum_{i=1}^n i \epsilon_i$ . Then we have

$$s(a) = \frac{\mathbb{P}(\sum_{i=1}^n (2i) Z_{2i} = n - a)}{\max_{\ell} \mathbb{P}(\sum_{i=1}^n (2i) Z_{2i} = \ell)} = \frac{\mathbb{P}\left(\sum_{i=1}^{(n-a)/2} i Z_i = \frac{n-a}{2}\right)}{\max_{\ell} \mathbb{P}\left(\sum_{i=1}^{\ell} i Z_i = \ell\right)} = \frac{p\left(\frac{n-a}{2}\right) x^{(n-a)/2} \prod_{i=1}^{(n-a)/2} (1 - x^i)}{\max_{\ell} p(\ell) x^{\ell} \prod_{i=1}^{\ell} (1 - x^i)}, \quad (7)$$

where  $p(n)$  denotes the number of integer partitions of  $n$ , which is an extremely well-studied sequence; see for example [1] and the references therein. In addition, we have  $\mathcal{L}(B|(a, B) \in E)$  is equivalent to the original problem with  $n$  replaced by  $(n - a)/2$ , hence the description of this algorithm as self-similar. In terms of random sampling of integer partitions, this is also within a constant factor of the fastest possible algorithm due to the efficient computation of  $p(n)$ ; see [1, Theorem 3.5]. More importantly, however, is that this type of divide-and-conquer is inherently probabilistic, table-free, and can be applied to other structures which are modeled by geometric random variables, namely, contingency tables.

Next, we revisit and generalize an approach in [26] for random sampling of nonnegative integer-valued  $(r, c)$ -contingency tables inspired by the above application for integer partitions. The generalization fixes an arbitrary set of entries to be 0, which doesn't change the approximation heuristic; however, it does a priori drastically change the computational complexity of computing the exact sampling rejection probability.

We start with a well-known probabilistic model for the entries in a random contingency table, generalized to include entries which are forced to be 0. In this section, we let  $W$  be any given  $m \times n$  matrix with values in  $\{0, 1\}$ . For each  $j = 1, 2, \dots, n$ , we define  $J_j := \{1, \dots, n\} \setminus \{i : W(i, j) = 1\}$ . Similarly, for each  $i = 1, 2, \dots, m$ , we define  $I_i := \{1, \dots, n\} \setminus \{j : W(i, j) = 1\}$ . Also, we let  $h_j = \sum_{i=1}^m W(i, j)$  denote the number of entries forced to be zero in column  $j$ , for  $j = 1, 2, \dots, n$ .

**Lemma 2.4.** *Let  $\mathbb{X}_W = (X_{ij})_{1 \leq i \leq m, 1 \leq j \leq n}$  denote a collection of independent geometric random variables with parameters  $p_{ij}$ , such that  $W(i, j) = 1$  implies  $X_{i,j}$  is conditioned to have value 0. If  $p_{ij}$  has the form  $p_{ij} = 1 - \alpha_i \beta_j$ , then  $\mathbb{X}_W$  is uniform restricted to  $(r, c)$ -contingency tables with zeros in entries indicated by  $W$ .*

*Proof.* Let  $\mathbb{X} = (X_{ij})_{1 \leq i \leq m, 1 \leq j \leq n}$  denote a collection of independent geometric random variables with parameters  $p_{ij}$ , where  $p_{ij}$  has the form  $p_{ij} = 1 - \alpha_i \beta_j$ . We have

$$\mathbb{P}(\mathbb{X} = \xi) = \prod_{i,j} \mathbb{P}(X_{ij} = \xi_{ij}) = \prod_{i,j} (\alpha_i \beta_j)^{\xi_{ij}} (1 - \alpha_i \beta_j) = \prod_i \alpha_i^{r_i} \prod_j \beta_j^{c_j} \prod_{i,j} (1 - \alpha_i \beta_j).$$

Since this probability does not depend on  $\xi$ , it follows that the restriction of  $\mathbb{X}$  to  $(r, c)$ -contingency tables is uniform. As the collection of random variables are independent, conditioning on any  $X_{i,j} = 0$  only changes the constant of proportionality, and does not affect the dependence on the  $\xi$ , hence

$$\mathbb{P}(\mathbb{X}_W = \xi) = \prod_{i,j: W(i,j)=0} \mathbb{P}(X_{ij} = \xi_{ij}) = \prod_{i,j: W(i,j)=0} (\alpha_i \beta_j)^{\xi_{ij}} (1 - \alpha_i \beta_j) = \prod_i \alpha_i^{r_i} \prod_j \beta_j^{c_j} \prod_{i,j} (1 - \alpha_i \beta_j);$$



i.e., it follows that the restriction of  $\mathbb{X}_W$  to  $(r, c)$ -contingency tables with forced zero entries indicated by  $W$  is uniform.  $\square$

**Lemma 2.5.** *Suppose  $\mathbb{X}_W$  is a collection of independent geometric random variables, where  $X_{i,j}$  has parameter  $p_{ij} = \frac{m-h_j}{m-h_j+c_j}$ , for all pairs  $(i, j)$  such that  $W(i, j) = 0$ , and  $p_{i,j} = 1$  for all pairs  $(i, j)$  such that  $W(i, j) = 1$ . Then the expected column sums of  $\mathbb{X}_W$  are  $c_1, c_2, \dots, c_n$ , and the expected row sums are  $\sum_{j \in I_1} \frac{c_j}{m-h_j}, \dots, \sum_{j \in I_m} \frac{c_j}{m-h_j}$ .*

*Proof.* Note first that  $\mathbb{E}X_{i,j} = p_{i,j}^{-1} - 1$ . Then for any  $j = 1, 2, \dots, n$ ,

$$\sum_{i=1}^m \mathbb{E}X_{i,j} = \sum_{i \in J_j} \frac{m-h_j+c_j}{m-h_j} - 1 = c_j$$

and similarly for any  $i = 1, 2, \dots, m$ ,

$$\sum_{j=1}^n \mathbb{E}X_{i,j} = \sum_{j \in I_i} \frac{m-h_j+c_j}{m-h_j} - 1 = \sum_{j \in I_i} \frac{c_j}{m-h_j}. \quad \square$$

Next, we describe the PDC algorithm and how the heuristic is utilized.

Suppose we have matrices  $W$  and  $\mathcal{O}$  with entries in  $\{0, 1\}$ . For any set of row sums and column sums  $(r, c)$ , let  $\Sigma(r, c, \mathcal{O}, W)$  denote the number of nonnegative integer-valued  $(r, c)$ -contingency tables with entry  $(i, j)$  forced to be even if the  $(i, j)$ th entry of  $\mathcal{O}$  is 1, and entry  $(i, j)$  forced to be 0 if the  $(i, j)$ th entry of  $W$  is 1. Let  $\mathcal{O}_{i,j}$  denote the matrix which has entries with value 1 in the first  $j-1$  columns, and entries with value 1 in the first  $i$  rows of column  $j$ , and entries with value 0 otherwise.

The algorithm we champion for contingency tables starts by considering the first entry in the first column not forced to be 0, say at entry  $(s, 1)$ , and denote the least significant bit by  $\epsilon_{s,1}$ . As we are in the first column, we have  $\mathcal{L}(\epsilon_{s,1}|E) = \mathcal{L}\left(\text{Bern}\left(\frac{q_1}{1+q_1}\right) \middle| E\right)$ , where  $E$  is given in Equation (5). In lieu of sampling from this conditional distribution directly, and a more practical alternative is to sample from  $\mathcal{L}\left(\text{Bern}\left(\frac{q_1}{1+q_1}\right)\right)$  and reject the outcome with probability proportional to  $1 - \mathbb{P}(E|\epsilon_{s,1})$ .

We define  $r_i(k) = (r_1, \dots, r_i - k, \dots, r_m)$  and  $c_j(k) = (c_1, \dots, c_j - k, \dots, c_n)$  for  $k \in \{0, 1\}$ . By Lemma 2.4, we have

$$\mathbb{P}(E|\epsilon_{s,1} = k) = \Sigma(r_s(k), c_1(k), W, \mathcal{O}_{i,j}) \cdot \left(q_1^{c_1-k}(1-q_1^2)\right) (1-q_1)^{m-h_1} \prod_{j=2}^n q_j^{c_j} (1-q_j)^{m-h_j}.$$

Then, since we reject *in proportion* to this probability, we normalize by all terms which do not depend on  $k$ , which gives

$$\mathbb{P}(E|\epsilon_{s,1} = k) \propto \Sigma(r_s(k), c_1(k), W, \mathcal{O}_{i,j}) \cdot q_1^{-k}, \quad (8)$$

where we use the notation  $a(k) \propto b(k)$  to mean that  $a(k)/b(k)$  is equal to a positive constant independent of  $k$  (but potentially depending on all other parameters). Once we accept a value for  $\epsilon_{s,1}$ , we then move to the next bit further down in the column and sample analogously, continuing in this manner column by column, left to right.

The following is a probabilistic formulation of the *exact* rejection probability for an arbitrary entry  $(i, j)$ , which assumes that all bits above the entry in the current column have already been sampled, as well as all bits in all columns to the left of  $(i, j)$ ; see Section 2 for the complete description of each random variable:

$$f(i, j, k, r, c, W) \propto \mathbb{P} \left( \begin{array}{lll} \sum_{\ell \in J_1}^{\ell < j} 2\xi''_{1,\ell}(q_\ell^2, c_\ell) & + \eta'_{1,j,i}(q_j, c_j) & + \sum_{\ell \in J_1}^{\ell > j} \xi'_{1,j}(q_\ell, c_\ell) & = r_1 \\ \sum_{\ell \in J_2}^{\ell < j} 2\xi''_{2,\ell}(q_\ell^2, c_\ell) & + \eta'_{2,j,i}(q_j, c_j) & + \sum_{\ell \in J_2}^{\ell > j} \xi'_{2,j}(q_\ell, c_\ell) & = r_2 \\ \vdots & & & \\ \sum_{\ell \in J_{i-1}}^{\ell < j} 2\xi''_{i-1,\ell}(q_\ell^2, c_\ell) & + \eta'_{i-1,j,i}(q_j, c_j) & + \sum_{\ell \in J_{i-1}}^{\ell > j} \xi'_{i-1,j}(q_\ell, c_\ell) & = r_{i-1} \\ \sum_{\ell \in J_i}^{\ell < j} 2\xi''_{i,\ell}(q_\ell^2, c_\ell) & + \eta''_{i,j,i}(q_j, c_j) & + \sum_{\ell \in J_i}^{\ell > j} \xi'_{i,j}(q_\ell, c_\ell) & = r_i - k \\ \sum_{\ell \in J_{i+1}}^{\ell < j} 2\xi''_{i+1,\ell}(q_\ell^2, c_\ell) & + \eta''_{i+1,j,i}(q_j, c_j) & + \sum_{\ell \in J_{i+1}}^{\ell > j} \xi'_{i+1,j}(q_\ell, c_\ell) & = r_{i+1} \\ \vdots & & & \\ \sum_{\ell \in J_m}^{\ell < j} 2\xi''_{m,\ell}(q_\ell^2, c_\ell) & + \eta''_{m,j,i}(q_j, c_j) & + \sum_{\ell \in J_m}^{\ell > j} \xi'_{m,j}(q_\ell, c_\ell) & = r_m \end{array} \right) \quad (9)$$

$$\times \mathbb{P} \left( \sum_{\ell \in I_i, \ell \leq i} 2\xi_{\ell,j}(q_j^2) + \sum_{\ell \in I_i, \ell > i} \xi_{\ell,j}(q_\ell) = c_j - k \right).$$

If we could evaluate the multivariate probability above exactly, or to some arbitrarily defined precision, then the bit-by-bit sampling approach would yield an exact sampling algorithm. However, we champion the following approximation to Equation (9) based on the heuristic that the dependencies between the random variables are strongest along the  $i$ -th row and the  $j$ -th column:

$$F(i, j, k, r, c, W) := \mathbb{P} \left( \sum_{\ell \in I_i, \ell < j} 2\xi''_{i,\ell}(q_\ell^2, c_j, \epsilon) + 2\eta''_{i,j,i} + \sum_{\ell \in I_i, \ell > j} \xi'_{i,\ell}(q_\ell) = r_i - k \right) \quad (10)$$

$$\times \mathbb{P} \left( \sum_{\ell \in J_j, \ell \leq i} 2\xi_{\ell,j}(q_j^2) + \sum_{\ell \in J_j, \ell > i} \xi_{\ell,j}(q_j) = c_j - k \right), \quad k \in \{0, 1\}.$$

Note that while this approximation does not encode the essential global information, it nonetheless enforces several necessary conditions like parity of the row and column, and is fast to compute directly.

## 2.3 An heuristic for Latin squares

The following is a variation of the motivating example of the previous section. Let  $0 < x < 1$ , and let  $X_1(x), X_2(x), \dots, X_n(x)$  denote independent Bernoulli random variables with  $X_i \equiv X_i(x)$  having distribution which is Bernoulli with parameter  $\frac{x^i}{1+x^i}$ ,  $i = 1, 2, \dots, n$ . Conditional on  $\sum_{i=1}^n i X_i(x) = n$ , the random variables  $X_i(x)$ ,  $i = 1, 2, \dots, n$ , represent the number of parts of size  $i$  in a uniformly chosen integer partition of size  $n$  into *distinct part sizes*; see [39, 22]. As before, suppose we wish to sample from  $\mathcal{L}((X_1(x), \dots, X_n(x)) | \sum_{i=1}^n i X_i(x) = n)$ . We then consider the following “spatial” PDC division

$$A = (X_1, X_3, X_5, \dots), \quad B = (X_2, X_4, X_6, \dots).$$

The result is again a self-similar, recursive PDC algorithm, with the rejection function given by

$$s(a) = \frac{\mathbb{P}(\sum_{i=1}^{n/2} (2i) X_{2i} = n - a)}{\max_{\ell} \mathbb{P}(\sum_{i=1}^{n/2} (2i) X_{2i} = \ell)} = \frac{p_d\left(\frac{n-a}{2}\right) x^{(n-a)/2} \prod_{i=1}^{(n-a)/2} (1+x^i)^{-1}}{\max_{\ell} p_d(\ell) x^{\ell} \prod_{i=1}^{\ell} (1+x^i)^{-1}}, \quad (11)$$

where  $p_d(n)$  is the number of integer partitions into distinct part sizes, which can be computed efficiently via, e.g., [43].

Let  $LS_n$  denote the set of Latin squares of order  $n$ , and let  $\mathcal{BC}_{n,m}$  denote the set of all  $n \times n$  binary contingency tables with line sums  $m$ . Consider any Latin square of order  $n$ , and consider the map  $\varphi_n : LS_n \rightarrow \mathcal{BC}_{n, \lceil n/2 \rceil}$  which replaces each entry having value  $x$  with  $x \bmod 2$ , i.e., each entry becomes a 1 if it is odd and 0 if it is even. The result is an  $n \times n$  binary contingency table with row sums and column sums  $\lceil n/2 \rceil$ . Let us assume as before that  $n$  is a perfect power of 2 for simplicity of exposition.

By symmetry, each entry in a Latin square is equally likely to be even or odd, subject to the condition that there are an equal number of odd and even entries in each row and column. It is thus natural to conjecture that the joint distribution of the parity of entries in a random Latin square of order  $n$  is close in some sense to a joint distribution of  $n \times n$  i.i.d. Bernoulli random variables with parameter  $1/2$ , subject to an equal number of 0s and 1s in each row and column; i.e., a uniform distribution on the corresponding set of binary contingency tables  $\mathcal{BC}_{n, n/2}$ . In fact, the standard approach to sampling from binary contingency tables, given in [15], does in fact model each entry as independent subject to the constraints, with entry  $(i, j)$  as having a Bernoulli distribution with parameter  $\frac{c_j}{m}$ , where  $c_j$  is the column sum and  $m$  is the number of rows; taking  $c_j = n/2$  and  $m = n$ , we obtain our aforementioned natural heuristic for the parity of entries in a Latin square of order  $n$ .

We may continue this reasoning for tables with prescribed entries forced to be zero, which is the inspiration for Algorithm 6. We similarly have analogous lemmas as in the previous section specifically for binary-valued tables, which are also straightforward generalizations from the work in [15], and so we omit the proof.

**Lemma 2.6.** *Let  $\mathbb{X}_W = (X_{ij})_{1 \leq i \leq m, 1 \leq j \leq n}$  denote a collection of independent Bernoulli random variables with parameters  $p_{ij}$ , such that  $W(i, j) = 1$  implies  $X_{i,j}$  is conditioned to have value 0. If  $p_{ij}$  has the form  $p_{ij} = 1 - \alpha_i \beta_j$ , then  $\mathbb{X}_W$  is uniform restricted to  $(r, c)$ -binary contingency tables with zeros in entries indicated by  $W$ .*

**Lemma 2.7.** *Suppose  $\mathbb{X}_W$  is a collection of independent Bernoulli random variables, where  $X_{i,j}$  has parameter  $p_{ij} = \frac{c_j}{m-h_j}$ , for all pairs  $(i, j)$  such that  $W(i, j) = 0$ , and  $p_{i,j} = 1$  for all pairs  $(i, j)$  such that  $W(i, j) = 1$ . Then the expected column sums of  $\mathbb{X}_W$  are  $c_1, c_2, \dots, c_n$ , and the expected row sums are  $\sum_{j \in I_1} \frac{c_j}{m-h_j}, \dots, \sum_{j \in I_m} \frac{c_j}{m-h_j}$ .*

There are thus two distinct places where bias can occur in our approach championed in Section 1.3. The first comes from the locations of the even/odd entries; even if we suppose that we may sample from  $\mathcal{BC}_{n, n/2}$  uniformly at random, there is no a priori reason to believe that  $\varphi_n$  maps the same number of Latin squares of order  $n$  to each element of  $\mathcal{BC}_{n, n/2}$ , and indeed this is easily seen to *not* be the case for  $n = 5$ , even though the distribution is close to uniform. However, supposing we did know precisely the multiset of counts for the image of  $\varphi_n$ , a natural sampling approach would be to generate samples from  $\mathcal{BC}_{n, n/2}$  uniformly and apply an appropriate rejection proportional to this multiset of counts (followed by the analogous procedure for the follow-up tables in the

algorithm); however, sampling uniformly from  $\mathcal{BC}_{n,n/2}$ , is not always straightforward or efficient, which motivates the following approximation algorithm.

In [15], the algorithm championed for binary contingency tables keeps track of an importance sampling weight, which represents the bias introduced by the sampling algorithm. The algorithm proceeds in the same manner as the one we champion for contingency tables, sampling entries column by column from top to bottom. Our suggested modification for binary contingency tables is to ignore the use of the Poisson-binomial distribution altogether, but still apply a rejection which captures several necessary conditions, along with what we suspect is the dominant source of bias in the sampling algorithm. That is, letting  $X_{i,j}(p_{i,j})$  denote a Bernoulli random variable with parameter  $p_j \equiv p_{i,j} = \frac{c_j}{n}$ , we propose a rejection probability of the form

$$B(i, j, k, r, c, W) := \mathbb{P} \left( \sum_{\ell \in I_i, \ell > j} X_{i,\ell}(p_\ell) = r_i - k \right) \times \mathbb{P} \left( \sum_{\ell \in J_j, \ell > i} X_{\ell,j}(p_j) = c_j - k \right), \quad k \in \{0, 1\}. \quad (12)$$

It is plausible that for the random sampling of Latin squares the two sources of bias at times cancel each other out, or potentially exacerbate the individual biases; it would certainly be interesting to explore this in more detail.

**Remark 2.8.** One can also ask the following fundamental question: Is the range of the injection  $\varphi_n$  equal to  $\mathcal{BC}_{n, \lceil n/2 \rceil}$  for all  $n$ ? In other words, do there exist binary contingency tables which, if generated, doom our approach from the start? The unfortunate answer is yes, found by exhaustive search for  $n = 7$ , although in practice we have not found there to be very many. The next natural question, of interest in complexity theory, would be if there are necessary and sufficient conditions on  $\mathcal{BC}_{n, \lceil n/2 \rceil}$  which indicate in advance whether such a configuration of even/odd entries cannot be completed. One could also ask similar questions pertaining to the later phases of the algorithm.

## 3 Algorithms

### 3.1 Nonnegative integer-valued contingency tables with prescribed 0 entries

For any  $1 \leq i \leq m$ ,  $1 \leq j \leq n$ , recall the definition of  $\Sigma(r, c, \mathcal{O}_{i,j}, W)$  from the previous section. For  $k = 0, 1$ , let  $r_i(k) = (r_1, \dots, r_i - k, \dots, r_m)$ , and  $c_j(k) = (c_1, \dots, c_j - k, \dots, c_n)$ . Then define

$$A_{i,j}(k) := \Sigma(r_i(k), c_j(k), \mathcal{O}_{i,j}, W),$$

$$g(i, j, k, r, c, W) := \frac{A_{i,j}(k)}{A_{i,j}(0) + A_{i,j}(1)}, \quad k \in \{0, 1\}. \quad (13)$$

The following notations and definitions from [26] are used in the algorithms.

- Geo( $q$ ) Geometric distribution with probability of success  $1 - q$ , for  $0 < q < 1$ , with  $\mathbb{P}(\text{Geo}(q) = k) = (1 - q)q^k$ ,  $k = 0, 1, 2, \dots$ .
- NB( $m, q$ ) Negative binomial distribution with parameters  $m$  and  $1 - q$ , given by the sum of  $m$  independent Geo( $q$ ) random variables, with

$$\mathbb{P}(\text{NB}(m, q) = k) = \binom{m + k - 1}{k} (1 - q)^m q^k.$$

- U Uniform distribution on  $[0, 1]$ . We will also denote random variables from this distribution as  $U$  or  $u$ ; should be considered independent of all other random variables, including other instances of  $U$ .
- Bern( $p$ ) Bernoulli distribution with probability of success  $p$ . Similarly to  $U$ , we will also use it as a random variable.
- $\xi_{i,j}(q)$  Geo( $q$ ) random variables which are independent for distinct pairs  $(i, j)$ ,  $1 \leq i \leq m, 1 \leq j \leq n$ .
- $\xi'_{i,j}(q, c_j, W)$  Random variables which have distribution

$$\mathcal{L} \left( \xi_{i,j}(q) \middle| \sum_{\ell \in J_j} \xi_{\ell,j}(q) = c_j \right),$$

and are independent of all other random variables  $\xi_{i,\ell}(q)$  for  $\ell \neq j$ .

- $2\xi''_{i,j}(q, c_j, W)$  Random variables which have distribution

$$\mathcal{L} \left( 2\xi_{i,j}(q^2) \middle| \sum_{\ell \in J_j} 2\xi_{\ell,j}(q^2) = c_j \right)$$

and are independent of all other random variables  $\xi_{i,\ell}(q)$  for  $\ell \neq j$ .

- $\eta'_{i,j,s}(q, c_j)$  Random variables which have distribution

$$\mathcal{L} \left( \xi_{i,j}(q) \middle| \sum_{\ell \in J_j, \ell \leq s} 2\xi_{\ell,j}(q^2) + \sum_{\ell \in J_j, \ell > s} \xi_{\ell,j}(q) = c_j \right),$$

and are independent of all other random variables  $\xi_{i,\ell}(q)$  for  $\ell \neq j$ .

- $2\eta''_{i,j,s}(q, c_j)$  Random variables which have distribution

$$\mathcal{L} \left( 2\xi_{i,j}(q^2) \middle| \sum_{\ell \in J_j, \ell \leq s} 2\xi_{\ell,j}(q^2) + \sum_{\ell \in J_j, \ell > s} \xi_{\ell,j}(q) = c_j \right),$$

and are independent of all other random variables  $\xi_{i,\ell}(q)$  for  $\ell \neq j$ .

- q** The vector  $(q_1, \dots, q_n)$ , where  $0 < q_i < 1$  for all  $i = 1, \dots, n$ .

Consider next the following subroutines to generate a random bit:

---

Exact sampling of least significant bit of entry  $(i, j)$  via enumeration of contingency tables

---

```

1: procedure EXACT( $i, j, r, c, A, W_b$ )
2:   if  $u \leq g(i, j, 0, r, c, W_b)$  then
3:     return 0
4:   else
5:     return 1
6:   end if
7: end procedure

```

---



---

Exact sampling of least significant bit of entry  $(i, j)$  via exact rejection sampling

---

```

1: procedure EXACT_PROBABILISTIC( $i, j, r, c, A, W_b$ )
2:   if  $u \leq f(i, j, 0, r, c, W_b)$  then
3:     return 0
4:   else
5:     return 1
6:   end if
7: end procedure

```

---

The previous two procedures are computationally intensive to evaluate, since they a priori rely on the entire solution set, i.e., the set of all potential values of entries. We suggest the following approximation approach, in which the  $(i, j)$ -th entry is decided based solely on the row sum  $r_i$  and column sum  $c_j$ . For each fixed  $1 \leq i \leq m$  and  $1 \leq j \leq n$ , let  $t_1, \dots, t_\nu$  denote the set of entries and the corresponding value which have a deterministic value (either the entire entry or the least significant bit) determined by the line sum conditions after the sampling of the least significant bit of entry  $(i, j)$ , and let  $r_{i,j}(t_1, \dots, t_\nu)$  and  $c_{i,j}(t_1, \dots, t_\nu)$  denote the residual row and column sums, respectively, given these entries. For example, if there is only one fillable entry in a given row or column, we may uniquely specify its value; if a row sum or column sum is zero, set all fillable entries in that row or column to zero; if there are two entries left in a given row and the row sum is even (and non-zero), then either both least significant bits are 0 or both are 1.

---

Given entries with values to fill in, return the set of determined entries.

---

```

1: procedure DETERMINISTIC_FILL( $\{t_1, \dots, t_\nu\}, r, c, A, W$ )
2:   Fill in the values  $\{t_1, \dots, t_\nu\}$  into the table  $A$ , update  $r, c$ , and  $W$  accordingly.
3:   For each row, fill in  $A$  any uniquely determined values based on the row sum.
4:   For each column, fill in  $A$  any uniquely determined values based on the column sum.
5:   If any values were filled in from the previous two steps, label them as  $\{t_{\nu+1}, \dots, t_d\}$  and
   update the values of  $r, c, A$ , and  $W$  accordingly.
6:   if no new elements are filled in, i.e.,  $d = 0$ , or if  $W$  is the matrix of all 1s then
7:     return  $(\{t_1, \dots, t_\nu, \dots, t_d\}, r, c, A, W)$ 
8:   else
9:     return Deterministic_Fill( $\{t_1, \dots, t_d\}, r, c, A, W$ )
10:  end if
11: end procedure

```

---

The rejection procedure we thus champion for approximate sampling of contingency tables is as follows. For each entry  $(i, j)$ , we calculate the set of all values which would be determined if we set that particular entry's bit equal to 0 or 1 using Procedure **Deterministic\_Fill**. We then sample the bit in proportion to the probability that all of the determined bits were also generated with their uniquely determined values, and apply a rejection probability similar to Equation (9), except we condition on these determined values. This is summarized in Procedure **Approximate\_Probabilistic** below.

---

Approximate sampling of least significant bit of entry  $(i, j)$  via rejection sampling

---

```

1: procedure APPROXIMATE_PROBABILISTIC( $i, j, r, c, A, W_b$ )
2:    $(\mathbf{s}_0, r_0, c_0, A_0, W_0) \leftarrow \text{Deterministic\_Fill}(\{(i, j, 0)\}, r, c, A, W)$ 
3:    $(\mathbf{s}_1, r_1, c_1, A_1, W_1) \leftarrow \text{Deterministic\_Fill}(\{(i, j, 1)\}, r, c, A, W)$ 
4:    $p_0 \leftarrow \prod_{\ell} \mathbb{P}(X_{i_{\ell}, j_{\ell}}^0 = k_{\ell}^0) \times F(i, j, 0, r_0, c_0, W_0)$ 
5:    $p_1 \leftarrow \prod_{\ell} \mathbb{P}(X_{i_{\ell}, j_{\ell}}^1 = k_{\ell}^1) \times F(i, j, 1, r_1, c_1, W_1)$ 
6:   if  $u \leq \frac{p_0}{p_0 + p_1}$  then
7:     return 0
8:   else
9:     return 1
10:  end if
11: end procedure

```

---

Note that the 0 in the term  $F(i, j, 0, r_1, c_1, W_1)$  above is *not* a typo, since the row sums and column sums are assumed to be updated in the **Deterministic\_Fill** algorithm. In addition, we did not specify whether the random variables  $X_{i,j}$  refer to the entire entry or the least significant bit, since they reflect the information contained in the  $t_{\ell}$ ,  $\ell = 1, \dots, \nu$ . In principle, the  $X_{i,j}$  could refer to any statistic related to entry  $(i, j)$ , though for us we only consider the value of the entry  $(i, j)$  or its least significant bit.

**Lemma 3.1.** *The time to evaluate Procedure **Deterministic\_Fill** is  $O(n^2 + m^2)$ .*

*Proof.* Each time we call the function, it iterates through all elements of the table, say twice, at cost  $O(mn)$ . At worst, we fill in one entire row or column at each recursive call of the function. Since we thus invoke the recursion at most  $\max(n, m)$  times, the result follows.  $\square$

We now state the main algorithm championed for contingency tables, which depending on several parameters determines the cost of the algorithm and the bias. This algorithm serves as the main inspiration for Algorithm 6, and is interesting in its own right.



---

**Algorithm 4** Random sampling of nonnegative integer-valued  $(r, c)$ -contingency table with forced zero entries in  $W$ .

---

```

1: Fix a procedure to sample a bit, denoted by Bit_Sampler.
2:  $A \leftarrow 0^{m \times n}$  (i.e.,  $A$  is initialized to be the  $m \times n$  matrix of all 0s).
3:  $(s, r, c, A, W) \leftarrow \text{DeterministicFill}(\{\}, r, c, A, W)$ .
4:  $b_{\max} \leftarrow \lceil \log_2(n) \rceil$ .
5: for  $b = 0, 1, \dots, b_{\max}$  do
6:    $W_b \leftarrow W$ .
7:    $X \leftarrow 0^{m \times n}$ .
8:   for  $j = 1, 2, \dots, n$  do
9:     for  $i = 1, 2, \dots, m$  do
10:      if  $W_b(i, j) \neq 1$  then
11:         $X_{i,j} \leftarrow \text{Bit\_Sampler}(i, j, r, c, W_b)$ .
12:         $(s, r, c, X, W_b) \leftarrow \text{DeterministicFill}(\{(i, j, X_{i,j})\}, r, c, X, W_b)$ .
13:      end if
14:    end for
15:  end for
16:   $A \leftarrow A + 2^b X$ 
17: end for
18: return  $A$ .
```

---

The function we choose for **Bit\_Sampler** determines the bias as well as the complexity, and we have presented three explicit possibilities: **Exact**, **Exact\_Probabilistic**, and **Approximate\_Probabilistic**. The rejection function  $g$  in Line 2 of Procedure **Exact**, given in Equation (13), is stated as an expression involving counting the total number of such tables; this is the direct analog of the recursive method, and typically requires a large table of values computed via a recursion. It is similarly difficult to compute the rejection function  $f$  in Line 2 of Procedure **Exact\_Probabilistic**, i.e., the probabilistic equivalent of  $g$ , but which motivates our approximation heuristic. Procedure **Approximate\_Probabilistic** offers an alternative which is both practical and an heuristically reasoned approximation for the exact quantities.

**Example 3.2.** As an example, an application of [26, Algorithm 2] yielded the following set of random bits in each of the five iterations of the algorithm for the random sampling of a contingency table with row sums  $(40, 30, 30, 50, 100, 50)$  and column sums  $(50, 50, 50, 50, 50, 50)$ .<sup>2</sup>

---

<sup>2</sup>An implementation in Mathematica took about 1 second.

$$\begin{pmatrix} 12 & 12 & 3 & 0 & 9 & 4 \\ 2 & 7 & 0 & 11 & 5 & 5 \\ 2 & 9 & 10 & 0 & 4 & 5 \\ 6 & 1 & 24 & 3 & 14 & 2 \\ 18 & 2 & 12 & 32 & 16 & 20 \\ 10 & 19 & 1 & 4 & 2 & 14 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \end{pmatrix} + 2^1 \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 \end{pmatrix} \\
+ 2^2 \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix} + 2^3 \begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \\
+ 2^4 \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} + 2^5 \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

**Example 3.3.** Consider the following example, a  $3 \times 4$  nonnegative integer-valued  $(r, c)$ -contingency table with row sums  $r = (20, 14, 18, 27)$  and column sums  $c = (13, 56, 10)$ . Furthermore, we force entries  $(1, 3)$ ,  $(2, 1)$ ,  $(3, 2)$ ,  $(3, 3)$ ,  $(3, 4)$  to be 0. The first step is Line 9, which transforms the initial sampling problem into the following:

$$\begin{array}{ccc}
\begin{array}{|c|c|c|c|} \hline & & \blacksquare & \\ \hline \blacksquare & & & \\ \hline & \blacksquare & \blacksquare & \blacksquare \\ \hline \end{array} & \begin{array}{l} 10 \\ 56 \\ 13 \end{array} & \longrightarrow \\
20 & 14 & 18 & 27
\end{array}
\longrightarrow
\begin{array}{|c|c|c|c|} \hline 7 & & \blacksquare & \\ \hline \blacksquare & & 18 & \\ \hline 13 & \blacksquare & \blacksquare & \blacksquare \\ \hline \end{array}
\begin{array}{l} 10 \\ 56 \\ 13 \end{array}
\longrightarrow$$

$$\begin{array}{|c|c|c|c|} \hline \blacksquare & & \blacksquare & \\ \hline \blacksquare & & \blacksquare & \\ \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \end{array}
\begin{array}{l} 3 \\ 38 \\ 0 \end{array}
\longrightarrow
\begin{array}{|c|c|} \hline & \\ \hline & \\ \hline \end{array}
\begin{array}{l} 3 \\ 38 \end{array}$$

$$\begin{array}{ccc}
0 & 14 & 0 & 27 \\
14 & & & 27
\end{array}$$

Now since this reduces to a standard  $2 \times 2$  contingency table, we can apply a standard algorithm or continue with Algorithm 4. We suggest in particular the *exact* sampling algorithm for  $2 \times n$  tables given in [26, Algorithm 3], which for this particular example would sample a uniform number in the set  $\{0, 1, 2, 3\}$  and then uniquely fill in the remaining entries.

### 3.2 Binary contingency tables

One can formulate an exact sampling rejection function for binary contingency tables as in the previous section, although the approximation algorithm is again the one we champion due to practical computing reasons. It is straightforward but notationally messy to adapt and write out

completely the analogous exact sampling procedures in this setting, and so we focus solely on our suggested approximate sampling approach.

Let  $B_j(q)$ ,  $j = 1, \dots, n$ , denote independent Bernoulli random variables with success probability  $1 - q$ . We have

$$b(n, \mathbf{q}, k) := \mathbb{P} \left( \sum_{j=1}^n B_j(q_j) = k \right) = \frac{1}{n+1} \sum_{\ell=0}^n C^{-\ell k} \prod_{j=1}^n \left( 1 + (C^\ell - 1)(1 - q_j) \right),$$

where  $C = \exp \left( \frac{2\pi i}{n+1} \right)$  is a  $(n+1)$ th root of unity. The expression in (14) is a numerically stable way to evaluate the convolution of a collection of independent Bernoulli random variables using a fast Fourier transform, see for example [34]. Next, we consider an  $m \times n$  matrix  $W$  which contains values in  $\{0, 1\}$ . A value 1 in entry  $(i, j)$  of  $W$  implies that this entry is forced to be zero. Given an index set  $J \subset \{1, 2, \dots, n\}$ , we define similarly

$$b(n, \mathbf{q}, k, J) := \mathbb{P} \left( \sum_{j \in J} B_j(q_j) = k \right).$$

For each  $i = 1, 2, \dots, m$ , let  $I_i \leftarrow \{1, \dots, n\} \setminus \{j : W(i, j) = 1\}$  denote the set of entries in row  $i$  which can be non-zero; and for each  $j = 1, 2, \dots, n$ , let  $J_j \leftarrow \{1, \dots, m\} \setminus \{i : W(i, j) = 1\}$  denote the set of entries in column  $j$  which can be non-zero. Also let  $\mathbf{q} = (c_1/(m-h_1), c_2/(m-h_2), \dots, c_n/(m-h_n))$  and let  $\nu_j = (c_j/(m-h_j), c_j/(m-h_j), \dots, c_j/(m-h_j))$ . Then we define for  $i \in I_i$  and  $j \in J_j$

$$H(i, j, k, r, c, W) := b(m, \mathbf{q}, r_i - k, I_i) \times b(n, \nu_j, c_j - k, J_j), \quad (14)$$

and 0 otherwise. We therefore define the following procedure to generate bits, noting that the wording of the `DeterministicFill` function defined in the previous section is equally applicable in this setting.

---

Approximate sampling of entry  $(i, j)$  of a binary contingency table

---

```

1: procedure APPROXIMATE_PROBABILISTIC_BINARY( $i, j, r, c, A, W$ )
2:    $(s_0, r_0, c_0, A_0, W_0) \leftarrow \text{DeterministicFill}(\{(i, j, 0)\}, r, c, A, W)$ 
3:    $(s_1, r_1, c_1, A_1, W_1) \leftarrow \text{DeterministicFill}(\{(i, j, 1)\}, r, c, A, W)$ 
4:    $p_0 \leftarrow \prod_{\ell} \mathbb{P}(X_{i_{\ell}, j_{\ell}}^0 = k_{\ell}^0) \times H(i, j, 0, r_0, c_0, W_0)$ 
5:    $p_1 \leftarrow \prod_{\ell} \mathbb{P}(X_{i_{\ell}, j_{\ell}}^1 = k_{\ell}^1) \times H(i, j, 1, r_1, c_1, W_1)$ 
6:   if  $u \leq \frac{p_0}{p_0 + p_1}$  then
7:     return 0
8:   else
9:     return 1
10:  end if
11: end procedure

```

---

**Lemma 3.4.** *Procedure `ApproximateProbabilisticBinary` has an arithmetic cost of  $O(n^2 + m^2)$*

*Proof.* By Lemma 3.1, each of the two calls to Procedure `DeterministicFill` take at most  $O(n^2 + m^2)$ . In addition, the cost to evaluate  $H$  via Equation (14) and Equation (14) is also at most  $O(n^2 + m^2)$ .  $\square$

We thus define the algorithm for approximate sampling of binary  $(r, c)$ -contingency tables as follows.

---

**Algorithm 5** Generation of an approximately uniform binary  $(r, c)$ -contingency table with forced zero entries in  $W$ .

---

```

1: Input:  $r = (r_1, \dots, r_m)$ ,  $c = (c_1, \dots, c_n)$ , and  $W \in \{0, 1\}^{m \times n}$ .
2: Output: binary  $(r, c)$ -contingency table with entries in  $W$  forced to be 0.
3:  $X \leftarrow 0^{m \times n}$  (i.e.,  $X$  is initialized to be the  $m \times n$  matrix of all 0s).
4:  $(s, r, c, X, W) \leftarrow \text{Deterministic\_Fill}(\{\}, r, c, X, W)$ .
5: for  $j = 1, 2, \dots, n$  do
6:   for  $i = 1, 2, \dots, m$  do
7:     if  $W(i, j) \neq 1$  then
8:        $X_{i,j} \leftarrow \text{ApproximateProbabilisticBinary}(i, j, r, c, W)$ .
9:        $(s, r, c, X, W) \leftarrow \text{Deterministic\_Fill}(\{(i, j, X_{i,j})\}, r, c, X, W)$ .
10:    end if
11:  end for
12: end for
13: return  $X$ .
```

---

### 3.3 Latin squares of order $n$

We now state a complete version of our main algorithm for random sampling of Latin squares.

---

**Algorithm 6** Generation of an approximately uniformly random Latin square of order  $n$ .

---

```

1:  $r \leftarrow (\lfloor n/2 \rfloor, \dots, \lfloor n/2 \rfloor)$ .
2:  $c \leftarrow (\lfloor n/2 \rfloor, \dots, \lfloor n/2 \rfloor)$ .
3: Let  $W$  denote the  $n \times n$  matrix with all 0 entries.
4:  $t \leftarrow$  the output from Algorithm 5 with input  $(r, c, W)$ .
5: for  $i = 1, 2, \dots, \lceil \log_2(n) \rceil$  do
6:    $r \leftarrow \lfloor r/2 \rfloor$ .
7:    $c \leftarrow \lfloor c/2 \rfloor$ .
8:   for  $b = 0, 1, \dots, 2^i - 1$  do
9:     Let  $W_b(i, j)$  be 0 if  $t(i, j) = b$ , and 1 otherwise, for all  $1 \leq i \leq m, 1 \leq j \leq n$ .
10:     $t_b \leftarrow$  the output from Algorithm 5 with input  $(r, c, W_b)$ .
11:     $t \leftarrow t + 2^i t_b$ 
12:   end for
13: end for
14: return  $t + 1$ 
```

---

**Remark 3.5.** It is easy to adapt this algorithm for the random sampling of  $n^2 \times n^2$  Sudoku matrices, with an extra rejection due to the local block constraints; see for example [50] for the precise definition. One could start with Algorithm 6 and perform a hard rejection at each iteration until the binary contingency table generated also satisfies the block constraints, or one could adapt the rejection probability to more effectively target the block constraints.

### 3.4 Algorithmic costs and a word of caution

We are unable at present to formulate precise runtime estimates, only to say we have implemented the approximation algorithms in practice and they exhibit roughly the runtime estimates we show below, with one major caveat. Part of the difficulty in precisely costing the algorithms is that, for the exact sampling algorithms, they rely on computing numerical quantities for which no known polynomial time algorithm is known. For the approximation algorithms, we have not yet undertaken a full analysis of the bias, and in fact, we caution that the algorithm can potentially reach a state for which the algorithm cannot terminate due to unsatisfiable constraints. In those cases it is up to the practitioner whether the algorithm ought to halt altogether and restart, or whether some partially completed table should be salvaged; again, since we have not performed a detailed analysis of the bias in these algorithms, we have not explored this matter further, except to note that the algorithm is surprisingly effective when it avoids such unsalvageable states.

In each of the estimates below, we assume  $m$  denotes the number of rows and  $n$  denotes the number of columns in a contingency table, and that  $W$  is some  $m \times n$  matrix with a 1 in the  $(i, j)$ -th entry if the  $(i, j)$ -th entry of the contingency table is forced to be 0. We also define

$$R := \max(r_1, \dots, r_m), \quad C := \max(c_1, \dots, c_n), \quad M := \max(R, C).$$

For Algorithm 4, with Procedure `ApproximateProbabilistic` used for the `BitSampler` routine, it is easy to see that barring the encounter of a state which cannot be completed, the expected number of random bits required is  $O(nm \log(M))$ , one for each visit to each entry per bit-level. For the arithmetic cost, each call to Procedure `ApproximateProbabilistic` involves  $O(R \log R)$  arithmetic operations, since the evaluation of the function  $F$  can be performed in a numerically stable manner using fast Fourier transforms. In addition, each call to Procedure `DeterministicFill` involves  $O(m^2 + n^2)$  arithmetic operations by Lemma 3.1. Thus, this corresponds to an informal arithmetic cost which is  $O((m^2 + n^2)mnR \log(R) \log(M))$  in the best case scenario.

For Algorithm 5, it is easy to see that the expected number of random bits is  $O(nm)$ , with  $O((n^2 + m^2)mn)$  arithmetic operations, since we iterate through each of the  $nm$  entries at most once, each time calling Procedure `ApproximateProbabilisticBinary`.

Finally, for Algorithm 6, the expected number of random bits required is  $O(n^2 \log(n))$  with  $O(n^4 \log n)$  arithmetic operations.

## 4 Acknowledgements

The author would like to acknowledge helpful discussions with Alejandro Morales, Igor Pak, Richard Arratia, and James Zhao.

## References

- [1] Richard Arratia and Stephen DeSalvo. Probabilistic divide-and-conquer: a new exact simulation method, with integer partitions as an example. *Combinatorics, Probability and Computing*,

- 25(3):324–351, May 2016.
- [2] W. Baldoni-Silva, J. A. De Loera, and M. Vergne. Counting integer flows in networks. *Found. Comput. Math.*, 4(3):277–314, 2004.
  - [3] Alexander Barvinok. Brunn-Minkowski inequalities for contingency tables and integer flows. *Adv. Math.*, 211(1):105–122, 2007.
  - [4] Alexander Barvinok. Enumerating contingency tables via random permanents. *Combin. Probab. Comput.*, 17(1):1–19, 2008.
  - [5] Alexander Barvinok. Asymptotic estimates for the number of contingency tables, integer flows, and volumes of transportation polytopes. *International Mathematics Research Notices*, 2009(2):348–385, 2009.
  - [6] Alexander Barvinok. Asymptotic estimates for the number of contingency tables, integer flows, and volumes of transportation polytopes. *Int. Math. Res. Not. IMRN*, (2):348–385, 2009.
  - [7] Alexander Barvinok. What does a random contingency table look like? *Combinatorics, Probability and Computing*, 19(04):517–539, 2010.
  - [8] Alexander Barvinok and J. A. Hartigan. An asymptotic formula for the number of non-negative integer matrices with prescribed row and column sums. *Trans. Amer. Math. Soc.*, 364(8):4323–4368, 2012.
  - [9] Alexander Barvinok, Zur Luria, Alex Samorodnitsky, and Alexander Yong. An approximation algorithm for counting contingency tables. *Random Structures Algorithms*, 37(1):25–66, 2010.
  - [10] Edward A. Bender. The asymptotic number of non-negative integer matrices with given row and column sums. *Discrete Math.*, 10:217–223, 1974.
  - [11] Edward A Bender and E Rodney Canfield. The asymptotic number of labeled graphs with given degree sequences. *Journal of Combinatorial Theory, Series A*, 24(3):296–307, 1978.
  - [12] Ivona Bezáková, Nayantara Bhatnagar, and Eric Vigoda. Sampling binary contingency tables with a greedy start. *Random Structures & Algorithms*, 30(1-2):168–205, 2007.
  - [13] Richard A. Brualdi and Geir Dahl. Matrices of zeros and ones with given line sums and a zero block. *Linear Algebra Appl.*, 371:191–207, 2003.
  - [14] Richard A Brualdi and Geir Dahl. Constructing  $(0, 1)$ -matrices with given line sums and certain fixed zeros. In *Advances in Discrete Tomography and Its Applications*, pages 113–123. Springer, 2007.
  - [15] Yuguo Chen, Persi Diaconis, Susan P Holmes, and Jun S Liu. Sequential monte carlo methods for statistical analysis of tables. *Journal of the American Statistical Association*, 100(469):109–120, 2005.
  - [16] Fan R. K. Chung, Ronald L. Graham, and Shing-Tung Yau. On sampling with markov chains. *Random Structures and Algorithms*, 9(1-2):55–77, 1996.
  - [17] Charles J Colbourn. The complexity of completing partial latin squares. *Discrete Applied*

- Mathematics*, 8(1):25–30, 1984.
- [18] Mary Cryan and Martin Dyer. A polynomial-time algorithm to approximately count contingency tables when the number of rows is constant. *Journal of Computer and System Sciences*, 67(2):291–310, 2003.
  - [19] Mary Cryan, Martin Dyer, Leslie Ann Goldberg, Mark Jerrum, and Russell Martin. Rapidly mixing markov chains for sampling contingency tables with a constant number of rows. *SIAM Journal on Computing*, 36(1):247–278, 2006.
  - [20] Geir Dahl. Permutation matrices related to sudoku. *Linear Algebra and its Applications*, 430(8):2457–2463, 2009.
  - [21] Jesús A. De Loera, Raymond Hemmecke, Jeremiah Tauzer, and Ruriko Yoshida. Effective lattice point counting in rational convex polytopes. *J. Symbolic Comput.*, 38(4):1273–1302, 2004.
  - [22] Amir Dembo, Anatoly Vershik, and Ofer Zeitouni. Large deviations for integer partitions, 1998.
  - [23] Stephen DeSalvo. Probabilistic divide-and-conquer: deterministic second half. *arXiv preprint arXiv:1411.6698*, 2014.
  - [24] Stephen DeSalvo. Exact, uniform sampling of latin squares and sudoku matrices. *Algorithmica*, To appear.
  - [25] Stephen DeSalvo. Improvements to exact Boltzmann sampling using probabilistic divide-and-conquer and the recursive method. *Electronic Notes in Discrete Mathematics*, To appear.
  - [26] Stephen DeSalvo and James Y Zhao. Random sampling of contingency tables via probabilistic divide-and-conquer. *arXiv preprint arXiv:1507.00070*, 2015.
  - [27] Persi Diaconis and Anil Gangolli. Rectangular arrays with fixed margins. In *Discrete probability and algorithms (Minneapolis, MN, 1993)*, volume 72 of *IMA Vol. Math. Appl.*, pages 15–41. Springer, New York, 1995.
  - [28] Persi Diaconis and Laurent Saloff-Coste. Comparison theorems for reversible markov chains. *The Annals of Applied Probability*, pages 696–730, 1993.
  - [29] Persi Diaconis, Bernd Sturmfels, et al. Algebraic algorithms for sampling from conditional distributions. *The Annals of statistics*, 26(1):363–397, 1998.
  - [30] Philippe Duchon. Random generation of combinatorial structures: Boltzmann samplers and beyond. 12 2011.
  - [31] Philippe Duchon, Philippe Flajolet, Guy Louchard, and Gilles Schaeffer. Boltzmann samplers for the random generation of combinatorial structures. *Combin. Probab. Comput.*, 13(4-5):577–625, 2004.
  - [32] Martin Dyer and Catherine Greenhill. Polynomial-time counting and sampling of two-rowed contingency tables. *Theoretical Computer Science*, 246(1):265–278, 2000.



- [33] Martin Dyer, Ravi Kannan, and John Mount. Sampling contingency tables. *Random Structures and Algorithms*, 10(4):487–506, 1997.
- [34] Manuel Fernandez and Stuart Williams. Closed-form expression for the Poisson-binomial probability density function. *Aerospace and Electronic Systems, IEEE Transactions on*, 46(2):803–817, 2010.
- [35] George S Fishman. Counting contingency tables via multistage markov chain monte carlo. *Journal of Computational and Graphical Statistics*, 21(3):713–738, 2012.
- [36] Philippe Flajolet and Robert Sedgewick. *Analytic combinatorics*. Cambridge University Press, Cambridge, 2009.
- [37] Roberto Fontana. Fractions of permutations. an application to sudoku. *Journal of Statistical Planning and Inference*, 141(12):3697–3704, 2011.
- [38] Roberto Fontana. Random latin squares and sudoku designs generation. *arXiv preprint arXiv:1305.3697*, 2013.
- [39] Bert Fristedt. The structure of random partitions of large integers. *Transactions of the American Mathematical Society*, 337(2):703–735, 1993.
- [40] I. J. Good and J. F. Crook. The enumeration of arrays and a generalization related to contingency tables. *Discrete Math.*, 19(1):23–45, 1977.
- [41] Catherine Greenhill and Brendan D. McKay. Asymptotic enumeration of sparse nonnegative integer matrices with specified row and column sums. *Adv. in Appl. Math.*, 41(4):459–481, 2008.
- [42] Diane Hernek. Random generation of  $2 \times n$  contingency tables. *Random Structures & Algorithms*, 13(1):71–79, 1998.
- [43] Loo-keng Hua. On the number of partitions of a number into unequal parts. *Trans. Amer. Math. Soc.*, 51:194–201, 1942.
- [44] Mark L. Huber. *Perfect Simulation*. Chapman & Hall/CRC Monographs on Statistics & Applied Probability. Taylor & Francis, 2015.
- [45] Mark T Jacobson and Peter Matthews. Generating uniformly distributed random latin squares. *Journal of Combinatorial Designs*, 4(6):405–437, 1996.
- [46] Mark Jerrum, Alistair Sinclair, and Eric Vigoda. A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries. *J. ACM*, 51(4):671–697 (electronic), 2004.
- [47] Sergei V. Kerov and Anatol M. Vershik. The characters of the infinite symmetric group and probability properties of the Robinson-Schensted-Knuth algorithm. *SIAM J. Algebraic Discrete Methods*, 7(1):116–124, 1986.
- [48] Shuji Kijima and Tomomi Matsui. Polynomial time perfect sampling algorithm for two-rowed contingency tables. *Random Structures & Algorithms*, 29(2):243–256, 2006.

- [49] Ben J Morris. Improved bounds for sampling contingency tables. *Random Structures & Algorithms*, 21(2):135–146, 2002.
- [50] Paul K Newton and Stephen A DeSalvo. The shannon entropy of sudoku matrices. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, page rspa20090522. The Royal Society, 2010.
- [51] Albert Nijenhuis and Herbert S. Wilf. A method and two algorithms on the theory of partitions. *J. Combinatorial Theory Ser. A*, 18:219–222, 1975.
- [52] Albert Nijenhuis and Herbert S. Wilf. *Combinatorial algorithms*. Academic Press, Inc. [Harcourt Brace Jovanovich, Publishers], New York-London, second edition, 1978. For computers and calculators, Computer Science and Applied Mathematics.
- [53] Patrick Eugene O’Neil. Asymptotics and random matrices with row-sum and column-sum restrictions. *Bull. Amer. Math. Soc.*, 75:1276–1282, 1969.
- [54] Boris Pittel. Random set partitions: asymptotics of subset counts. *journal of combinatorial theory, Series A*, 79(2):326–359, 1997.
- [55] James Gary Propp and David Bruce Wilson. Exact sampling with coupled markov chains and applications to statistical mechanics. *Random structures and Algorithms*, 9(1-2):223–252, 1996.
- [56] George W. Soules. New permanental upper bounds for nonnegative matrices. *Linear Multilinear Algebra*, 51(4):319–337, 2003.
- [57] J. H. van Lint and R. M. Wilson. *A course in combinatorics*. Cambridge University Press, Cambridge, second edition, 2001.
- [58] A.M. Vershik. Statistical mechanics of combinatorial partitions, and their limit shapes. *Functional Analysis and Its Applications*, 30:90–105, 1996.
- [59] Darren J Wilkinson. Parallel bayesian computation. *Statistics Textbooks and Monographs*, 184:477, 2006.
- [60] Krasimir Yordzhev. On the number of disjoint pairs of s-permutation matrices. *Discrete Applied Mathematics*, 161(18):3072–3079, 2013.